PRISMA SHIELD

DYSON

DEEP LOGIC AUDIT REPORT

Dyson: Balancer and Dystopia Strategies

MAR 03 2023

PRISMA SHIELD

Table of Contents

Overview

This audit only covers the Dyson strategies based on Balancer and Dystopia. It does not cover any other strategies built by Dyson. The set of contracts this audit covers are:

- DysonBalancerVault
- DysonMaximizerBalancerVault
- StrategyBalancerAC
- MaximizerBalancer
- BalancerRouterUtils
- DysonDystopiaVault
- DysonMaximizerDystopiaVault
- StrategyDystopia
- MaximizerDystopia
- DystopiaRouterUtils
- DynamicFeeManager
- StratManagerUpgradeable
- StratManagerUpgradeableCommon

All of the above contracts are deployed as upgradeable proxies. This means that the code can be changed under the same contract addresses after the initial deployment. Therefore, this audit report only covers the initial code deployment. If the code is upgraded, this audit report would be out-of-date. Therefore, please ensure trust in the team before interacting with these contracts.

What is a Deep Logic Audit?

A deep logic smart contract audit is a human-driven code review that checks all of the code business logic for bugs, mathematical errors, and security risks. The audit verifies that the code honors the whitepaper. In addition, this service includes mainnet testing and proactive communication with the project owners to ensure full comprehension of the project to provide the best possible code review quality.

Overview

Findings Summary

		Findings Resolved
Total Findings	**30**	**30**
High Security Findings	3	3
Medium Security Findings	2	2
High Logical Findings	6	6
Medium Logical Findings	4	4
Informational Findings	15	15

ID	Section	Type	Severity	Page	Status
MBV-01	DysonMaximizerBalancerVault	Security	High	10	Resolved
MBV-02	DysonMaximizerBalancerVault	Security	High	11	Resolved
MBV-03	DysonMaximizerBalancerVault	Security	Medium	12	Resolved
MBV-04	DysonMaximizerBalancerVault	Logical	Informational	13	Resolved
DBV-01	DysonBalancerVault	Security	High	15	Resolved
DBV-02	DysonBalancerVault	Logical	Medium	16	Resolved
SBC-01	StrategyBalancerAC	Security	High	18	Resolved

Overview

Findings Summary

ID	Section	Type	Severity	Page	Status
SBC-02	StrategyBalancerAC	Logical	High	19	Resolved
SBC-03	StrategyBalancerAC	Logical	High	20	Resolved
SBC-04	StrategyBalancerAC	Logical	High	21	Resolved
SBC-05	StrategyBalancerAC	Security	Medium	22	Resolved
SBC-06	StrategyBalancerAC	Logical	Medium	23	Resolved
SBC-07	StrategyBalancerAC	Logical	Informational	24	Resolved
SBC-08	StrategyBalancerAC	Logical	Informational	25	Resolved
SBC-09	StrategyBalancerAC	Logical	Informational	26	Resolved
SBC-10	StrategyBalancerAC	Logical	Informational	27	Resolved
SBC-11	StrategyBalancerAC	Logical	Informational	28	Resolved
SBC-12	StrategyBalancerAC	Logical	Informational	29	Resolved
SBC-13	StrategyBalancerAC	Logical	Informational	30	Resolved
BRU-01	BalancerRouterUtils	Logical	High	32	Resolved
BRU-02	BalancerRouterUtils	Logical	High	33	Resolved
BRU-03	BalancerRouterUtils	Security	Medium	34	Resolved

Overview

Findings Summary

ID	Section	Type	Severity	Page	Status
BRU-04	BalancerRouterUtils	Logical	Informational	35	Resolved
BRU-05	BalancerRouterUtils	Logical	Informational	36	Resolved
BRU-06	BalancerRouterUtils	Logical	Informational	37	Resolved
BRU-07	BalancerRouterUtils	Logical	Informational	38	Resolved
MBR-01	MaximizerBalancer	Logical	Informational	40	Resolved
MBR-02	MaximizerBalancer	Logical	Informational	41	Resolved
SDA-01	StrategyDystopia	Logical	Informational	45	Resolved
MDA-01	MaximizerDystopia	Logical	Informational	47	Resolved

Contract Addresses

All of these contracts are deployed on the Polygon network.

Balancer Vaults

WMATIC/stMATIC

DysonBalancerVault: 0x7140e011fD54D7e31108C67e068220774f9769D6
StrategyBalancerAC: 0xD2D3DF150F0c11CaAf6B2ef45239e370c8B34Fc9

WMATIC/stMATIC-USDC/USDT

DysonMaximizerBalancerVault: 0x1Fe04A05AB3f0a9955Ca8bb1A65FdB73a781a3e9
MaximizerBalancer: 0xE38ba9cbcFbcCDfD5f1ecC271c5EebD23f680c2F
BalancerRouterUtils: 0xF6Ed68b87BD5728c94377fCeB9F8606abEDcA7c4

Dystopia Vaults

USDC/FRAX

DysonDystopiaVault: 0xA647566dBc1629dc7cbd04a7B845f0b726ee1F62
StrategyDystopia: 0x6353F4ac331C5108aC8dF2c86377fd2b5028e72c

USD+/USDC

DysonDystopiaVault: 0xE3520C23ce36C4429245E6e3112D80c724D12598
StrategyDystopia: 0x1b2d9d03BB5EC3077da545E26B1BDe4228c61e36

FXS/FRAX

DysonDystopiaVault: 0x5B133085dF488AeC4eA352d77349514a298801E2
StrategyDystopia: 0x84814A5276B13C5506B61eAEa4fD7C1A41B6Af3F

WMATIC/stMATIC

DysonDystopiaVault: 0xb200Bf7fb384482dF5Db372AF18b82EA5Bc6aA90
StrategyDystopia: 0x0a7A85787a88e1A3d1C471C59FbbF3Ac2834ECE1

USD+/CLAM

DysonDystopiaVault: 0xa6329dbc68ce5ed86abD2f759D5405B42E40103E
StrategyDystopia: 0x98FBf00E9D0a90d77B455f51CCcb5bF5fc84Bf60

USD+/SPHERE

DysonDystopiaVault: 0x4CCd0283B2aFa4C5bE745552fD166a941B8256e6
StrategyDystopia: 0x8ED79D0EEfDB631EBA40dc17B5E019212c03D036

Contract Addresses

WMATIC/MaticX

DysonDystopiaVault: 0x4E52beBf1F21179D42358fD08d483A6806D38CD8
StrategyDystopia: 0x7113e7519bAfF951D23b652532346eE66D03A9Bf

USDC/TUSD

DysonDystopiaVault: 0x50AF83c737e485151fAffc196d98c8CAafa851FE
StrategyDystopia: 0x5ab4C05B43B8ADC3344b5c7D7a27431b7B2319DA

FRAX/MAI

DysonDystopiaVault: 0x5C5fd212dcfC0e916299a562f94B434a6DB63B91
StrategyDystopia: 0x24DC093594CDD135952E5188AE621102c4555427

USD+/stMATIC

DysonDystopiaVault: 0xac116Cea700Fb242436d63b10d06e2477EC6B9Fd
StrategyDystopia: 0x90A7a09942c0352Cd5be7A57FF09BD0EE28e59C1

USDC/FRAX-PEN/MATIC

DysonMaximizerDystopiaVault: 0xE937D9b81D6e46691Aa0Bdf9E63eaAB1ac218798
MaximizerDystopia: 0x99bc41105BAE7DC7A4EE19491B6B43fee502508a
DystopiaRouterUtils: 0x4393FBf83Baa26a1F6b22fa429187300A7D64cf7

USD+/USDC-PEN/MATIC

DysonMaximizerDystopiaVault: 0xCD41D5ba498D38F316816A0D4ED330236610659F
MaximizerDystopia: 0xb754D6919da9d61E8aB3Ef267b4703Cbc25F17F5
DystopiaRouterUtils: 0x030fCFA8443Ae304f83115a670C2b6AEf95C0E57

FXS/FRAX-PEN/MATIC

DysonMaximizerDystopiaVault: 0x055d03d9Bd1409aDA6F8E22487DC9B47457d061c
MaximizerDystopia: 0x06226cA6DB58B6778C153b503ffbc5AA0EaCEed4
DystopiaRouterUtils: 0xe7C48b2F48EEBDAeE59F6d20501d2090ca25523a

WMATIC/stMATIC-PEN/MATIC

DysonMaximizerDystopiaVault: 0xcf9F4Bdfae24516aa9566B6D9F8a92cf24D9cf43
MaximizerDystopia: 0x9c9F193f4b7B3eA95d160c9b72b2837Af5E0D193
DystopiaRouterUtils: 0x26614830684734E5467FC792CAB6d9Ca78Da3419

Contract Addresses

USD+/CLAM-PEN/MATIC

DysonMaximizerDystopiaVault: 0xCDdBFe5621DF2557eE74A256A9b09e36721f42DB
MaximizerDystopia: 0x4e896C2440B66bF5B8D4dc0C8eD82BEca2Fac510
DystopiaRouterUtils: 0x072A719c08e4E732d660f5c2db5A813a6EF3255F

USD+/SPHERE-PEN/MATIC

DysonMaximizerDystopiaVault: 0xC69fd8176f784CC63Ef1ae7F764D0819b5bF2382
MaximizerDystopia: 0x3FE076f00101d3350ca28f12d46c0Fb3fAF4AC0b
DystopiaRouterUtils: 0xCd905449C57ac403c850a08B5B9eA697A57b2333

WMATIC/MaticX-PEN/MATIC

DysonMaximizerDystopiaVault: 0xc312dAB28AA16ce67DFeF14970E997628209cBD7
MaximizerDystopia: 0xb7E695AcEFF82e8090EC88BC607c1213B2a1465A
DystopiaRouterUtils: 0x3BFf2034F5417B39bfe8720D6eA92D559Bf0F4dA

USDC/TUSD-PEN/MATIC

DysonMaximizerDystopiaVault: 0x7ff9976f59763a9e794bD8A9841948Fda7a46AeD
MaximizerDystopia: 0xeEA6E4B65ED174882A628ba7f324BF2fC5e3049b
DystopiaRouterUtils: 0x56607f967b4110d93090d3eCB50Eb914321713C0

FRAX/MAI-PEN/MATIC

DysonMaximizerDystopiaVault: 0x6eaE69C5Fdb3f736015DECb5E75f90762e41D6a3
MaximizerDystopia: 0xb4e038523F65b0850207BDED53bE2351D07b693B
DystopiaRouterUtils: 0xb4d3D29d1a1cf7F650f96B54CA335D4E4068BD55

USD+/stMATIC-PEN/MATIC

DysonMaximizerDystopiaVault: 0xeC562fE650E44551299C645f7CA0817796380E44
MaximizerDystopia: 0x1428013Bdcd45f1689C80fcD5a8Eb8fA1C4E73c8
DystopiaRouterUtils: 0x090d2EACBa3BEBE4f0B34CFC6C2Ef2039466c718

Audit Findings

DysonMaximizerBalancerVault

MBV-01 - Security High Severity

The setStrategy function does not set _isStrategyInitialized to true, allowing the strategy contract attached to the vault to change, which could lock out users' funds.

Recommendation

Set _isStrategyInitialized to true in setStrategy.

Resolution

The team has implemented the recommendation.

Audit Findings

DysonMaximizerBalancerVault

MBV-02 - Logical High Severity

The earn function is public, and can be used by users maliciously with any arbitrary amount for the _shares argument to reset the rewards claimed.

Recommendation

The function should be changed to private, or it should not receive a _shares argument and instead calculate the shares within the function itself.

Resolution

The team has implemented the recommendation.

The function has been changed to internal.

Audit Findings

DysonMaximizerBalancerVault

MBV-03 - Logical Medium Severity

In deposit, 1000 wei of want tokens are being sent to the dead address when there are 0 deposited funds. Upon asking the team the intention behind this, they said that the aim was to prevent inflation attacks, where a malicious user could be the initial depositer in the vault and would deposit a very small amount, thereby receiving a small amount of shares, and then they would send a lot more want tokens to the strategy without using the deposit function, which would result in a small amount of shares having a very large value of want tokens, thereby "inflating" the value of the shares, which would be undesirable. Unfortunately, the burning of the 1000 wei of want tokens does not prevent these kinds of inflation attacks, and just results in want tokens being wasted unnecessarily.

Recommendation

Don't take into consideration want tokens sent to the strategy outside of the deposit function, and optionally build a simple mechanism to only allow a limited amount of want tokens to be deposited without shares being issued (donations) if this is a required feature. Moreover, if a trusted user is the initial depositer in the vault, and they deposit a large enough amount of want tokens (without ever removing that deposit), this would deter inflation attacks.

Resolution

The team has implemented the recommendation.

Any want tokens that are sent to the strategy outside of the deposit function are ignored.

Audit Findings

DysonMaximizerBalancerVault

MBV-04 - Logical Medium Severity

In deposit and withdraw, when calling afterDepositAndWithdraw, the code uses balanceOf(msg.sender) instead of balanceBelongTo(msg.sender) which is what MaximizerBalancer uses to calculate the already-claimed reward amounts.

Recommendation

Use balanceBelongTo(msg.sender) in afterDepositAndWithdraw.

Resolution

The team has implemented the recommendation.

Overview

DysonMaximizerBalancerVault

- This contract is the vault that users interact with directly to deposit and withdraw their funds for the MaximizerBalancer contract.

- The want publicly viewable function returns the address of the LP token that users deposit to and withdraw from this vault.

- The deposit, depositAll, withdraw, and withdrawAll functions are the ones that users interact with to deposit and withdraw their want tokens.

- The strategy publicly viewable variable contains the address of the MaximizerBalancer contract. The contract owner has admin powers to set this value only once.

- The boostPool publicly viewable variable contains the address of the BoostPoolBalancer contract. This contract is unaudited by Prisma Shield at the current time. The Dyson team has promised that the code path using this contract will be disabled until the contract is audited. The contract owner has admin powers to change this value.

- The receipt tokens that users receive upon depositing in this vault are non-transferrable, except by the boostPool contract.

- The inCaseTokensGetStuck function can be used to extract stuck ERC20 tokens that are not want from the contract. The contract owner has admin powers to call this function.

Audit Findings

DysonBalancerVault

DBV-01 - Security High Severity

The setStrategy function does not set _isStrategyInitialized to true, allowing the strategy contract attached to the vault to change, which could lock out users' funds.

Recommendation

Set _isStrategyInitialized to true in setStrategy.

Resolution

The team has implemented the recommendation.

Audit Findings

DysonBalancerVault

DBV-02 - Logical Medium Severity

The setStrategy function does not have the check require(address(strategy_.vault()) == address(this)), which would render the contract useless, and it would need to be redeployed.

Recommendation

Add the check require(address(strategy_.vault()) == address(this)).

Resolution

The team has implemented the recommendation.

Overview

DysonBalancerVault

- This contract is the vault that users interact with directly to deposit and withdraw their funds for the StrategyBalancerAC contract.

- The want publicly viewable function returns the address of the LP token that users deposit to and withdraw from this vault.

- The deposit, depositAll, withdraw, and withdrawAll functions are the ones that users to interact with to deposit and withdraw their want tokens.

- The strategy publicly viewable variable contains the address of the StrategyBalancerAC contract. The contract owner has admin powers to set this value only once.

- The inCaseTokensGetStuck function can be used to extract stuck ERC20 tokens that are not want from the contract. The contract owner has admin powers to call this function.

Audit Findings

StrategyBalancerAC

SBC-01 - Security High Severity

The recoverFunds and recoverAllFunds functions can be used by the contract owner to access user funds.

Recommendation

These functions should be deleted and replaced by inCaseTokensGetStuck that would only allow accessing tokens that are not used by this contract.

Resolution

The team has implemented the recommendation.

Audit Findings

StrategyBalancerAC

SBC-02 - Logical High Severity

If reward1 == native, then chargeFees will result in double-taxation. This is because generalFeeOnProfits calculates the fee on the amount of reward1 after it has increased following the swap from reward2 to native in the _harvest function.

Recommendation

chargeFees should receive as an argument the amount of reward1 before the swap from reward2 to native, and use that to calculate the fee.

Resolution

The team has implemented the recommendation.

The code has been generalized, and reward1 is no longer allowed to be the same as native.

Audit Findings

StrategyBalancerAC

SBC-03 - Logical High Severity

If reward2 == native, this could result in unexpected behaviour.

Recommendation
Add require(native != reward2) in _StrategyBalancerAC_init_unchained.

Resolution
The team has implemented the recommendation.

The team has also added an additional check to not allow reward1 to be the same as reward2.

Audit Findings

StrategyBalancerAC

SBC-04 - Logical High Severity

addLiquidity has IERC20Upgradeable(reward1).balanceOf(address(this)) - nativeHalf instead of IERC20Upgradeable(native).balanceOf(address(this)) - nativeHalf which would result in mathematical errors.

Recommendation

Change the code to IERC20Upgradeable(native).balanceOf(address(this)) - nativeHalf.

Resolution

The team has implemented the recommendation.

Audit Findings

StrategyBalancerAC

SBC-05 - Security Medium Severity

In swap and swapUniswap, it might be worth adding require(IERC20Upgradeable(route[route.length - 1]).balanceOf(address(this)) > balanceBefore) to ensure that the balance of the token being received has increased. (balanceBefore = route[route.length - 1]).balanceOf(address(this)) at the beginning of the function before any swaps took place).

Recommendation

Add the check require(IERC20Upgradeable(route[route.length - 1]).balanceOf(address(this)) > balanceBefore) to swap and swapUniswap.

Resolution

The team has implemented the recommendation.

Audit Findings

StrategyBalancerAC

SBC-06 - Logical Medium Severity

harvestOnDeposit is initialized to false, and withdrawalFee is initialized to 0. This contradicts setHarvestOnDeposit, which sets withdrawalFee to 10 if harvestOnDeposit is set to false.

Recommendation

Either change the initializations of harvestOnDeposit and withdrawalFee to match the logic of setHarvestOnDeposit, or change the logic of setHarvestOnDeposit to match the initializations of harvestOnDeposit and withdrawalFee.

Resolution

The team has implemented the recommendation.

withdrawalFee is being initialized to 10.

Audit Findings

StrategyBalancerAC

SBC-07 - Logical Informational Severity

The addLiquidity function uses reward1Half for both swaps, which could result in some reward1 tokens not being swapped due to integer division.

Recommendation

Use IERC20Upgradeable(reward1).balanceOf(address(this)) - reward1Half for one of the two swaps, to ensure that all of reward1 is used.

Resolution

The team has implemented the recommendation.

Audit Findings

StrategyBalancerAC

SBC-08 - Logical Informational Severity

Supplying deadline to swapUniswap is not required and is a waste of gas, because that function is called from the same contract, where block.timestamp is constant.

Recommendation

Remove the deadline argument from swapUniswap. and use block.timestamp or any number larger than that directly where required.

Resolution

The team has implemented the recommendation.

Audit Findings

StrategyBalancerAC

SBC-09 - Logical Informational Severity

In the swap, addLiquidity, and swapUniswap functions, the slippage check should not be required, as the swap itself is happening within the same block as the price check, and so slippage is not a factor that needs to be considered in this case. Slippage only needs to be considered when someone is getting the price at a different block from the swap itself (for example, when someone is checking the price in the UI of a DEX before swapping). The slippage control implementation is meaningless as it will always pass due to the price calculated always being exactly equal to the actual amount of tokens eventually received.

Recommendation

Remove the slippage code to save gas.

Resolution

The team has implemented the recommendation.

Audit Findings

StrategyBalancerAC

SBC-10 - Logical Informational Severity

In chargeFees, if the sum of the individual fees could potentially not be equal to generalFeeAmount due to integer division, which would result in some fees not being transferred.

Recommendation

Add the following piece of code:

```
if (callFeeAmount + feeAmount1 + feeAmount2 +
    strategistFeeAmount != generalFeeAmount) {
  if (fee1 > 0) {
    feeAmount1 = (generalFeeAmount - callFeeAmount -
                          feeAmount2 - strategistFeeAmount) ;
  } else if (fee2 > 0) {
    feeAmount2 = (generalFeeAmount - callFeeAmount -
                          feeAmount1 - strategistFeeAmount) ;
  } else {
    strategistFeeAmount= (generalFeeAmount - callFeeAmount -
                                       feeAmount1 - feeAmount2) ;
  }
}
```

Resolution

The team has implemented the recommendation.

Audit Findings

StrategyBalancerAC

SBC-11 - Logical Informational Severity

In addLiquidity, there is no need to use address(assets[j]), just simply doing assets[j] == lpToken0 and assets[j] == lpToken1 is enough because assets[j] is already of type address.

Recommendation

Remove the cast of assets[j] to address.

Resolution

The team has implemented the recommendation.

Audit Findings

StrategyBalancerAC

SBC-12 - Logical Informational Severity

In addLiquidity, request is being set twice.

Recommendation

Set request only once.

Resolution

The team has implemented the recommendation.

Audit Findings

StrategyBalancerAC

SBC-13 - Logical Informational Severity

In _harvest, it is better to check nativeBalanceAfter > nativeBalanceBefore instead of nativeBalanceAfter - nativeBalanceBefore > 0, which saves a bit of gas and avoids unexpected smart contract errors.

Recommendation

Change the code to nativeBalanceAfter > nativeBalanceBefore.

Resolution

The team has implemented the recommendation.

Overview

StrategyBalancerAC

- This contract implements the basic autocompounding strategy based on Balancer, which deposits the want LP tokens into Balancer, and periodically harvests the rewards, swaps them to the want LP tokens, and deposits them to Balancer. Some fees are taken from the harvested rewards for different purposes.

- The feeOnProfits publicly viewable variable contains the percentage taken from profits as fees. It defaults to 4%, and can be set to a maximum of 10%. This fee is split between fee1, fee2, callFee, and strategistFee, which respectively default to 65%, 35%, 0%, and 0%. These fees are respectively sent to the addresses feeRecipient1, feeRecipient2, strategist, and the address that created the harvest transaction (tx.origin) or the address specified in the callFeeRecipient argument. callFee can be set to maximum of 11.1% (of the feeOnProfits). The contract owner has admin powers to change these values.

- The withdrawalFee publicly viewable variable contains the percentage of withdrawn want tokens that are retained in the contract to be redeposited. This value defaults to 0.1% of the amount withdrawn, and can be set to a maximum of 0.5%. The contract owner has admin powers to change this value.

- The inCaseTokensGetStuck function can be used to extract stuck ERC20 tokens that are not any of tokens used by this contract. The contract owner has admin powers to call this function.

- The pause function can be used to disable new deposits. The panic function disables new deposits and withdraws all the want tokens from Balancer. The unpause function re-enables deposits and deposits all want tokens in the contract into Balancer. The contract owner has admin powers to call these function.

Audit Findings

BalancerRouterUtils

BRU-01 - Logical High Severity

zapPrimaryWantToNative and zapNativeToSecondaryWant transfer the zapped tokens to maximizer instead of to msg.sender. Unless these functions are only meant to be called from the maximizer contract, this does not look like it is working as intended. This is effectively taking funds from the user without reward or without depositing them properly. In addition, no trace token amounts of secondaryLpToken0 or secondaryLpToken1 are being sent back to the user in zapNativeToSecondaryWant.

Recommendation

If only the maximizer contract can call these functions, then a corresponding require statement should be added at the start of these functions. Or simply just send the tokens directly to msg.sender.

Resolution

The team has implemented the recommendation.

Only the maximizer contract can call these functions.

Audit Findings

BalancerRouterUtils

BRU-02 - Logical High Severity

exitPoolBalancer assumes that native is one of the tokens in the LP, but there is no guarantee of that, which could result in unexpected errors.

Recommendation

Add the statement statement require(native == lpToken0 || native == lpToken1) to _StrategyBalancerAC_init_unchained.

Resolution

The team has implemented the recommendation.

Audit Findings

BalancerRouterUtils

BRU-03 - Security Medium Severity

Same comment in swapBalancer and exitPoolBalancer as SBC-11 about adding a check that the balance of the token received has increased.

Recommendation

Add a check in swapBalancer and exitPoolBalancer that the balance of the token received has increased.

Resolution

The team has implemented the recommendation.

Audit Findings

BalancerRouterUtils

BRU-04 - Logical Informational Severity

Same comment as SBC-04 about the slippage checks not being required.

Recommendation

Remove the slippage checks.

Resolution

The team has implemented the recommendation.

Audit Findings

BalancerRouterUtils

BRU-05 - Logical Informational Severity

zapNativeToSecondaryWant does not need + 10 seconds in the addLiquidity call. Using block.timestamp alone is sufficient, because everything happens in the same block.

Recommendation

Only use block.timestamp in the addLiquidity call.

Resolution

The team has implemented the recommendation.

Audit Findings

BalancerRouterUtils

BRU-06 - Logical Informational Severity

In zapPrimaryWantToNative and zapNativeToSecondaryWant, why not just send the full token amounts to the maximizer contract instead of the difference between after and before? The reason is that there is no other way to extract funds that might accidentally be sent to BalancerRouterUtils, so they would be stuck there, which would be a waste, so might as well put them to work.

Recommendation

Send the full token amounts in zapPrimaryWantToNative and zapNativeToSecondaryWant to maximizer.

Resolution

The team has implemented the recommendation.

Audit Findings

BalancerRouterUtils

BRU-07 - Logical Informational Severity

In zapNativeToSecondaryWant, it is better to check secondaryWantBalanceAfter > secondaryWantBalanceBefore instead of secondaryWantBalanceAfter - secondaryWantBalanceBefore > 0, which saves a bit of gas and avoids unexpected smart contract errors.

Recommendation

Change the code to secondaryWantBalanceAfter > secondaryWantBalanceBefore.

Resolution

The team has implemented the recommendation.

Overview

BalancerRouterUtils

- This contract implements some utilities around token swaps that are used by MaximizerBalancer.

- The zapPrimaryWantToNative function is used to swap want tokens to native tokens, where native is one of the two tokens of the want LP token. This function can only be called by the MaximizerBalancer contract.

- The zapNativeToSecondaryWant function is used to swap native tokens to the secondaryWant LP tokens. This function can only be called by the MaximizerBalancer contract.

- The setMaximizer function is used to set the address of the MaximizerBalancer contract in the maximizer function. The contract owner has admin powers to call this function.

Audit Findings

MaximizerBalancer

MBR-01 - Logical Informational Severity

In _updateLP, why not just depositLpAndStake the full secondaryWant.balanceOf(address(this)) amount instead of secondaryWantBalanceAfter - secondaryWantBalanceBefore? That is to ensure all secondaryWant tokens in the contract are being put to work.

Recommendation

Use the full secondaryWant.balanceOf(address(this)) amount in depositLpAndStake.

Resolution

The team has implemented the recommendation.

Audit Findings

MaximizerBalancer

MBR-02 - Logical Informational Severity

Same comment in chargeFees as SBC-07 about ensuring that the sum of the individual fees is equal to generalFeeAmount.

Resolution

The team has implemented the recommendation.

Overview

MaximizerBalancer

- This contract implements the maximizer autocompounding strategy based on Balancer, which deposits the want LP tokens into Balancer through the DysonBalancerVault contract, and swaps the profits to the secondaryWant LP token and deposits it into Penrose to generate extra DYST and PEN token rewards. The secondaryWant, DYST, and PEN tokens can be claimed as rewards by the users. Some fees are taken from the harvested want token rewards for different purposes.

- The feeOnProfits publicly viewable variable contains the percentage taken from profits as fees. It defaults to 4%, and can be set to a maximum of 10%. This fee is split between fee1, fee2, callFee, and strategistFee, which respectively default to 65%, 35%, 0%, and 0%. These fees are respectively sent to the addresses feeRecipient1, feeRecipient2, strategist, and the address that created the harvest transaction (tx.origin) or the address specified in the callFeeRecipient argument. callFee can be set to maximum of 11.1% (of the feeOnProfits). The contract owner has admin powers to change these values.

- The withdrawalFee publicly viewable variable contains the percentage of withdrawn want tokens that are retained in the contract to be redeposited. This value defaults to 0.1% of the amount withdrawn, and can be set to a maximum of 0.5%. The contract owner has admin powers to change this value.

- The claimRewards function triggers a harvest, and sends to the caller any secondaryWant, DYST, and PEN reward tokens that belongs to them. This also happens whenever a user deposits or withdraws in the DysonMaximizerBalancerVault contract.

- The pause function can be used to disable new deposits. The panic function disables new deposits and withdraws all the deposited tokens from Balancer and Penrose. The unpause function re-enables deposits and deposits all tokens back into Balancer and Penrose.The contract owner has admin powers to call these functions.

Overview

DysonMaximizerDystopiaVault

- This contract is the vault that users interact with directly to deposit and withdraw their funds for the MaximizerDystopia contract.

- The want publicly viewable function returns the address of the LP token that users deposit to and withdraw from this vault.

- The deposit, depositAll, withdraw, and withdrawAll functions are the ones that users to interact with to deposit and withdraw their want tokens.

- The strategy publicly viewable variable contains the address of the MaximizerDystopia contract. The contract owner has admin powers to set this value only once.

- The boostPool publicly viewable variable contains the address of the BoostPool contract. This contract is unaudited by Prisma Shield at the current time. The Dyson team has promised that the code path using this contract will be disabled until the contract is audited. The contract owner has admin powers to change this value.

- The receipt tokens that users receive upon depositing in this vault are non-transferrable, except by the boostPool contract.

- The inCaseTokensGetStuck function can be used to extract stuck ERC20 tokens that are not want from the contract. The contract owner has admin powers to call this function.

Overview

DysonDystopiaVault

- This contract is the vault that users interact with directly to deposit and withdraw their funds for the StrategyDystopia contract.

- The want publicly viewable function returns the address of the LP token that users deposit to and withdraw from this vault.

- The deposit, depositAll, withdraw, and withdrawAll functions are the ones that users to interact with to deposit and withdraw their want tokens.

- The strategy publicly viewable variable contains the address of the StrategyDystopia contract. The contract owner has admin powers to set this value only once.

- The inCaseTokensGetStuck function can be used to extract stuck ERC20 tokens that are not want from the contract. The contract owner has admin powers to call this function.

Audit Findings

StrategyDystopia

SDA-01 - Logical Informational Severity

To be extra safe, in _giveAllowances, add IERC20Upgradeable(want).safeApprove(chef, 0); and IERC20Upgradeable(output).safeApprove(dystRouter, 0); before giving the full allowance.

Resolution

The team has implemented the recommendation.

Overview

StrategyDystopia

- This contract implements the basic autocompounding strategy based on Dystopia, which deposits the want LP tokens into Dystopia, and periodically harvests the rewards, swaps them to the want LP tokens, and deposits them to Dystopia. Some fees are taken from the harvested rewards for different purposes.

- The feeOnProfits publicly viewable variable contains the percentage taken from profits as fees. It defaults to 4%, and can be set to a maximum of 10%. This fee is split between fee1, fee2, callFee, and strategistFee, which respectively default to 65%, 35%, 0%, and 0%. These fees are respectively sent to the addresses feeRecipient1, feeRecipient2, strategist, and the address that created the harvest transaction (tx.origin) or the address specified in the callFeeRecipient argument. callFee can be set to maximum of 11.1% (of the feeOnProfits). The contract owner has admin powers to change these values.

- The withdrawalFee publicly viewable variable contains the percentage of withdrawn want tokens that are retained in the contract to be redeposited. This value defaults to 0.1% of the amount withdrawn, and can be set to a maximum of 0.5%. The contract owner has admin powers to change this value.

- The inCaseTokensGetStuck function can be used to extract stuck ERC20 tokens that are not any of tokens used by this contract. The contract owner has admin powers to call this function.

- The pause function can be used to disable new deposits. The panic function disables new deposits and withdraws all the want tokens from Dystopia. The unpause function re-enables deposits and deposits all want tokens in the contract into Dytopia. The contract owner has admin powers to call these functions.

Audit Findings

MaximizerDystopia

MDA-01 - Logical Informational Severity

Please make sure to remove the hardhat console import.

Resolution

The team has implemented the recommendation.

Overview

MaximizerDystopia

- This contract implements the maximizer autocompounding strategy based on Dystopia, which deposits the want LP tokens into Dystopia through the DysonDystopiaVault contract, and swaps the profits to the secondaryWant LP token and deposits it into Penrose to generate extra DYST and PEN token rewards. The secondaryWant, DYST, and PEN tokens can be claimed as rewards by the users. Some fees are taken from the harvested want token rewards for different purposes.

- The feeOnProfits publicly viewable variable contains the percentage taken from profits as fees. It defaults to 4%, and can be set to a maximum of 10%. This fee is split between fee1, fee2, callFee, and strategistFee, which respectively default to 65%, 35%, 0%, and 0%. These fees are respectively sent to the addresses feeRecipient1, feeRecipient2, strategist, and the address that created the harvest transaction (tx.origin) or the address specified in the callFeeRecipient argument. callFee can be set to maximum of 11.1% (of the feeOnProfits). The contract owner has admin powers to change these values.

- The withdrawalFee publicly viewable variable contains the percentage of withdrawn want tokens that are retained in the contract to be redeposited. This value defaults to 0.1% of the amount withdrawn, and can be set to a maximum of 0.5%. The contract owner has admin powers to change this value.

- The claimRewards function triggers a harvest, and sends to the caller any secondaryWant, DYST, and PEN reward tokens that belongs to them. This also happens whenever a user deposits or withdraws in the DysonMaximizerDystopiaVault contract.

- The pause function can be used to disable new deposits. The panic function disables new deposits and withdraws all the deposited tokens from Dystopia and Penrose. The unpause function re-enables deposits and deposits all tokens back into Dystopia and Penrose.The contract owner has admin powers to call these functions.

How to Interpret Findings

Security - High Severity

Indicates that users' funds are at risk or that there is a high probability of exploitation.

Security - Medium Severity

No risk to the protocol or those who interact with it, however it should be highlighted nonetheless.

Logical - High Severity

Indicates that the errors puts users' funds at risk, or can result in significant functional failure in the code.

Logical - Medium Severity

Indicates some functional failure or discrepancy in the code.

Logical - Informational

Minor discrepancy between the intended functionality of the code and the implementation, which does not result in functional failure, or a recommendation to improve the logic.

Yellow Text

Indicates centralization of control and admin powers.

Red Text

An important warning to take note of.

Disclaimer

The information in this deep logic audit report objectively describes the smart contracts being audited, and points out logical and mathematical errors, security risks and vulnerabilities, and optimization opportunities in the audited code. This deep logic audit does not ensure the correctness or authenticity of any software or dApp that interacts with or claims to interact with any smart contract.

This audit report does not constitute any advice whatsoever. You are solely responsible for conducting your own due diligence and consulting your financial advisor before making any investment decisions. While our deep logic audits raise the level of security, reliability, mathematical accuracy, and logical soundness of the smart contracts reviewed, they do not amount to any form of warranty or guarantee that the reviewed smart contracts are void of any weaknesses, vulnerabilities, or bugs. Prisma Shield and its founders, employees, owners, and associates are not liable for any damage or loss of funds. Please ensure trust in the team prior to investing as this deep logic audit does not guarantee the promised use of your funds.

You can find the full disclaimer, terms and conditions, and privacy policy on the prismashield.com website.

PRISMA SHIELD

Introducing Deep Logic Smart Contract Auditing to Web3

prismashield.com

prismashield@gmail.com

PrismaShield

PrismaShield